CPE 349: Assignment 1– Comparison Sorts

A comparison sort is a type of sorting algorithm that compares elements in a list (array, file, etc) using a comparison operation that determines which of two elements should occur first in the final sorted list. The operator usually satisfies the following:

1. $a \le a$ for all a in the set
2. if $a \le b$ and $b \le c$ then $a \le c$ (transitive)
3. if $a \le b$ and $b \le a$ then $a=b$ (anti-symmetric)
4. for all a and b, either $a \le b$ or $b \le a$ // any two items can be compared (makes it a total rather than a partial order)

In situations where #3 does not strictly hold then, it is possible that and b are in some way different and both $a \le b$ and $b \le a$; in this case either may come first in the sorted list. In a **stable sort**, the input order determines the sorted order in this case.

A metaphor for thinking about comparison sorts in the abstract is that someone has a set of unlabeled weights and a balance scale. Their goal is to line up the weights in order by their weight without any information except that obtained by placing two weights on the scale and seeing which one is heavier (or if they weigh the same).

The following link is helpful.

- Comparison Sorting Visualizations: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Your goal for this lab is to implement versions of Insertion Sort - **insertSort(anArr)**, Selection Sort – **selectSort(anArr),** and Merge Sort Sort - **mergeSort(anArr)** that will sort an array of integers and **count and return the number of comparisons**. Each function takes as input an array of integers, sorts the array while counting the comparisons at the same time, and returns the number of comparisons. After the function completes, **anArr** should be sorted.

Submit in a file called **Sorts.java** to PolyLearn.

The worst-case runtime complexity is $\Theta(n^2)$ for selection and insertion sort. Why? Be able to describe a worst case for each and write out the summation that represents the number of comparisons. Note: There is a fundamental limit on the smallest number of comparisons that comparison sort can achieve in the worst case, namely
$\Omega(n*\log n)$. We will discuss the reasons for this in class.

Deliverables: Before the due date and time you must submit two files to PolyLearn.

1. Class **Sorts.java** must contain the source code for the following methods
   public static long insertSort (int [] data)
   public static long selectSort (int [] data)
   public static long mergeSort (int [] data

The source code for the java class Sorts.java that performs sorts must be testable by the TA's. In order for your code to run is must meet the above interface exactly including the file name that contains the class. You are to hand in only the above methods in the Sorts.java file. Do not hand in any test data or testing methods.

- Make sure your program does not use "package" or any classes not in the standard library.
- The methods specified above should not print anything or write to a file. If they do they will not run in our test harness.
- Make sure that your source code is well written. e.g. variable names should be meaningful but not too long, white space should make it easy to see program flow, and comments should explain how and why the algorithm works. Comments should **not** just be a rewording of the java code!

2. A pdf file, **SortAnalysis.pdf**, that contains the following **clearly labeled 1.a, 1.b etc.**:
   1. A derivation using the methodology discussed in class to determine the **worst case** complexity of each algorithm.
   2. **Tables** that shows the number of comparisons computed by your program by counting the number of comparisons required for different data set sizes and types of data input: sorted, reverse sorted, random. The tables should be given in the order: Table 1 Selection Sort, Table 2: Insertion Sort, Table 3 Merge Short. Data set sizes are 10, 1000, 2,000, 3,000, 4,000. The **table rows** are labeled by the data set sizes 10, 1000, 2,000, 3,000, 4,000. The **table columns** are labeled by the type of data , e.g. sorted, etc
      a. Column 1 label: **sorted**
      b. Column 2 label: **revsorted**
      c. Column 3 label: **random**
   3. Finally, interpret the results in the table.
      a. Do they agree with your theoretical analysis?
      b. If they do not agree, give your explanation.